Analysis of Random Polling Dynamic Load Balancing

Peter Sanders
Lehrstuhl Informatik für Ingenieure und Naturwissenschaftler
University of Karlsruhe
D-76128 Karlsruhe (Germany)
E-mail: sanders@ira.uka.de

April 28, 1994

Abstract

Dynamic load balancing is crucial for the performance of many parallel algorithms. Random Polling, a simple randomized algorithm, has proved to be very efficient in practice for applications like parallel depth first search. This paper derives tight bounds for the scalability of Random Polling which are for the first time able to explain its superior performance analytically. In some cases, Random Polling even turns out to be optimal. The analysis is based on a fairly general model of the application and the parallel machine. Some of the proof-techniques used might also turn out be useful for the analysis of other parallel algorithms. Finally, a simple initialization scheme is presented which vastly improves the algorithm's performance during the startup phase.

1 Introduction

Load Balancing is one of the central issues in parallel computing. Since for many applications it is almost impossible to predict how much computation a given subproblem involves, a dynamic load balancing (DLB) strategy is necessary which is able to keep the processors busy without incurring an undue overhead.

DLB comes in many guises. This paper is concerned with a very simple yet important model of the problem domain. The only thing the load balancer knows about a piece of work is, whether it is exhausted or not. Furthermore, a piece of work can be split into two parts. But nothing is known about the relative size of the two parts or their interactions.

One application domain for which this is a useful model is depth first tree search. Search trees are often very irregular and the size of a subtree is hard to

predict, but it is easy to split the search stack into two parts. Also, interactions between the subtrees often follow the tree structure (e. g. reporting results) or they are hard to exploit by a load balancer anyway (e. g. broadcasting of the current best solution or accessing distributed hash tables). Note that depth first tree traversal is a central aspect of many AI and OR-applications and of parallel functional and logical programming languages.

This paper focuses on *Random Polling* (RP), a simple yet effective randomized DLB scheme. Every processing element (PE) works at exactly one piece of work. A PE whose piece of work is exhausted, polls randomly determined PEs until it finds one which is busy. The busy PE splits its piece of work and transmits one part to the idle PE.

After introducing the notation necessary for a formal treatment in Section 2, Section 3 gives references to related work. Section 4 presents some general results about probabilistic algorithm analysis which are quite interesting by themselves. Then Section 5 gives the analysis of RP. Section 6 describes an initialization scheme which is able to remove some inefficiencies of basic RP during the startup phase.

2 Definitions

The analysis considers a MIMD computer consisting of n identical PEs numbered 0 through $n-1$ which interact by exchanging messages through a network of diameter $d(n)$. The size w of a problem is measured in units of sequential execution time. A task (piece of work) can be represented by a message of size $s(w)$. When a task has been split $h(w)$ times it is assumed to have been reduced to some atomic size $g(w)$.

The performance of a DLB-scheme is assessed by analyzing a problem without interactions between subtasks where DLB is the only source of parallelization overhead. Natural performance measures are the parallel execution time $T_{\text{par}}(n, w)$ or the efficiency

$$E = \frac{w}{nT_{\text{par}}(n, w)} \tag{1}$$

But the complexity of discussing bivariate functions can be avoided by fixing E and solving equation 1 for w yielding the *isoefficiency function* $w(n)$. This function is a convenient measure for the degree of scalability of an algorithm. For a more detailed discussion of various scalability measures see [Kum90].

Since RP is a randomized algorithm, a meaningful analysis has to be a probabilistic one. Two different notions of probabilistic behavior turn out to be useful. One is the traditional notion of average case behavior:

Definition 1 *A random variable $X(n)$ (where n is a parameter) is in $O(f(n))$*

on the average *iff*

$$\mathbf{E}X(n) = \sum_i i\mathbf{P}\left[X(n) = i\right] = O(f(n))$$

Another notion is behavior *with high probability*[1] which is somewhat more complicated but often quite useful:

Definition 2 *A random variable* $X(n)$ *is in* $O(f(n))$ with high probability *or* $X(n) = \tilde{O}\left(f(n)\right)$ *for short iff*

$$\exists c > 0, n_0 > 0 : \forall \beta \geq 1, n \geq n_0 : \mathbf{P}\left[X(n) > c\beta f(n)\right] \leq n^{-\beta}$$

Section 4 makes a connection between these two notions. But in order to do that, two other terms from probability theory are needed:

Definition 3 $\tilde{x}_\alpha$ *is* α-quantile *of a random variable* X *iff*

$$\mathbf{P}\left[X < \tilde{x}_\alpha\right] \leq \alpha < \mathbf{P}\left[X \leq \tilde{x}_\alpha\right]$$

Definition 4 *A* conditional expected value *of a random variable* X *is*

$$\mathbf{E}[X|c] := \sum_i i\mathbf{P}\left[X|c\right]$$

3 Related Work

Due to its simplicity and effectiveness, Random Polling (the name is adopted from [KA91]) has probably been invented independently by several researchers (e.g. [FM87]). There is so much practical evidence for its effectiveness in a multitude of settings (see also [FMM91, PFK93, San94b, San94a]) that this paper concentrates on analysis.

In [KZ93] it is proved that for $d(n) = O(1)$ and $s(w), g(w) = O(1)$ RP has an isoefficiency function in $O(n^2 \log n)$ with high probability. Much tighter is the result in [KA91]: If $s(w), g(w) = O(1)$ and $h(w) = O(\log w)$, the isoefficiency function of RP is in $O(nd(n)\log^2 n)$ on the average. This already indicates a quite good scalability. But it falls short of explaining why RP is in practice more efficient than a deterministic algorithm introduced in the same paper which has an isoefficiency in $\Theta(nd(n)\log n)$.

Another randomized DLB algorithm is based on dynamic tree-embeddings into butterfly networks or hypercubes [L+89, Ran94]. For $s(w), g(w) = O(1)$ it has an isoefficiency function in $O(nh(n))$ with high probability which is asymptotically optimal. However, the algorithm has no notion of granularity control resulting in high memory requirements and a possibly quite small upper bound on the achievable efficiency due to communication overhead. Another interesting result from [L+89] is that no deterministic tree embedding with the same performance can exist.

[1] Various slightly different introductions of this notion can be found in the literature (e. g. [Raj92, Lei92]); this paper tries to use the strictest reasonable interpretation.

4 Some Basic Results

There are some fairly general results which greatly simplify the analysis of RP. Theorem 1 formulates a powerful property of behavior with high probability: A bound on the maximum of polynomially many random variables is given by the maximum of the individual bounds for the random variables. Theorem 2 cites a frequently used result about coin flipping experiments and Theorem 3 makes a connection between behavior with high probability and average case behavior.

Theorem 1 *Let $X_1(n) = \tilde{O}(f_1(n)), \ldots, X_m(n) = \tilde{O}(f_m(n))$ be random variables where m is at most polynomial in n. Then*

$$\max_{i=1}^{m} X_i(n) = \tilde{O}\left(\max_{i=1}^{m} f_i(n)\right)$$

Proof: Let $k > 0$ be a constant such that $m \leq n^k$ for sufficiently large n. Let $c = (k+1)\max_{i=1}^{m} c_i$ where c_i is the constant of proportionality used to show that $X_i(n) = \tilde{O}(f_i(n))$. We need to show that
$\mathbf{P}\left[\max_{i=1}^{m} X_i(n) > c\beta \max_{i=1}^{m} f_i(n)\right] \leq n^{-\beta}$ for any $\beta \geq 1$ for sufficiently large n.

$$\begin{aligned}
\mathbf{P}\left[\max_{i=1}^{m} X_i(n) > c\beta \max_{i=1}^{m} f_i(n)\right] &= \mathbf{P}\left[\bigcup_{j=1}^{m}\left(X_j(n) > c\beta \max_{i=1}^{m} f_i(n)\right)\right] \\
&\leq \sum_{j=1}^{m} \mathbf{P}\left[X_j(n) > c\beta \max_{i=1}^{m} f_i(n)\right] \\
&\leq m \max_{j=1}^{m} \mathbf{P}\left[X_j(n) > c\beta \max_{i=1}^{m} f_i(n)\right] \\
&\leq m \max_{j=1}^{m} \mathbf{P}\left[X_j(n) > c\beta f_j(n)\right] \\
&\leq m \max_{j=1}^{m} \mathbf{P}\left[X_j(n) > c_j(k+1)\beta f_j(n)\right] \\
&\leq m \max_{j=1}^{m} \mathbf{P}\left[X_j(n) > c_j(k+\beta)\, f_j(n)\right] \\
&\leq m n^{-(k+\beta)} \\
&\leq n^k n^{-(\beta+k)} \\
&= n^{-\beta}
\end{aligned}$$

□

This result has a special case of particular importance to parallel algorithms: The run time of a parallel algorithms is determined by the slowest PE. If the runtime of individual PEs is known with high probability Theorem 1 makes it easy to estimate the overall run time.

Theorem 2 *Let the random variable X represent the number of heads after n independent flips of a loaded coin where the probability for a head is p. Then*

$$\mathbf{P}\left[X \leq (1-\epsilon)np\right] \leq e^{-\epsilon^2 np/3} \text{ for } 0 < \epsilon < 1$$

[Che52, Raj92].

The above theorems are often useful for the analysis of parallel algorithms but they are not directly applicable if average case behavior is the measure of interest. However, it is often possible to infer the average case behavior once the behavior with high probability is known. One way to make this transition is based on the following lemma:

Lemma 1 *If $\tilde{x}_\alpha$ is* α-quantile *of a random variable X and it is known that $\mathbf{E}[X|X \geq \tilde{x}_\alpha] \leq \tilde{x}_\alpha + \mathbf{E}X$ then $\mathbf{E}X \leq \frac{\tilde{x}_\alpha}{\alpha}$.*

Proof: Splitting the sum defining $\mathbf{E}X$ at $\tilde{x}_\alpha$ yields:

$$\mathbf{E}X = \sum_i i\mathbf{P}\left[X = i\right] = \sum_{i<\tilde{x}_\alpha} i\mathbf{P}\left[X = i\right] + \sum_{i\geq\tilde{x}_\alpha} i\mathbf{P}\left[X = i\right] \tag{2}$$

The left part of the sum can be estimated using the definition of a quantile:

$$\sum_{i<\tilde{x}_\alpha} i\mathbf{P}\left[X = i\right] \leq \sum_{i<\tilde{x}_\alpha} \tilde{x}_\alpha\mathbf{P}\left[X = i\right] = \tilde{x}_\alpha \sum_{i<\tilde{x}_\alpha} \mathbf{P}\left[X = i\right] \leq \tilde{x}_\alpha\alpha$$

Also by definition of a quantile

$$\mathbf{P}\left[X = i|X \geq \tilde{x}_\alpha\right] = \frac{\mathbf{P}\left[X = i \cap X \geq \tilde{x}_\alpha\right]}{\mathbf{P}\left[X \geq \tilde{x}_\alpha\right]} \geq \frac{\mathbf{P}\left[X = i\right]}{1-\alpha}$$

for $i \geq \tilde{x}_\alpha$ and, $\mathbf{P}\left[X = i|X \geq \tilde{x}_\alpha\right] = 0$ for $i < \tilde{x}_\alpha$. The right part of the sum in quation 2 can be massaged to exploit this knowledge:

$$\begin{aligned}
\sum_{i\geq\tilde{x}_\alpha} i\mathbf{P}\left[X = i\right] &= (1-\alpha)\sum_{i\geq\tilde{x}_\alpha} i\frac{\mathbf{P}\left[X = i\right]}{1-\alpha} \\
&\leq (1-\alpha)\sum_{i\geq\tilde{x}_\alpha} i\mathbf{P}\left[X = i|X \geq \tilde{x}_\alpha\right] \\
&= (1-\alpha)\sum_i i\mathbf{P}\left[X = i|X \geq \tilde{x}_\alpha\right] \\
&\leq (1-\alpha)(\tilde{x}_\alpha + \mathbf{E}X)
\end{aligned}$$

Putting the pieces together gives a relation which can be solved for $\mathbf{E}X$.

$$\begin{aligned}
\mathbf{E}X &\leq \tilde{x}_\alpha\alpha + (1-\alpha)(\tilde{x}_\alpha + \mathbf{E}X) = \tilde{x}_\alpha + (1-\alpha)\mathbf{E}X \\
\mathbf{E}X &\leq \frac{\tilde{x}_\alpha}{\alpha}
\end{aligned}$$

□

Now it is simple to prove the following theorem:

Theorem 3 *If $X(n) = O(f(n))$ with high probability and $\mathbf{E}[X(n)|X(n) \geq \tilde{x}(n)_{1-\frac{1}{n}}] \leq \tilde{x}(n)_{1-\frac{1}{n}} + \mathbf{E}X(n)$ then $X(n) = O(f(n))$ on the average.*

Proof: Setting $\beta = 1$ in Definition 2 yields that there is a c such that for sufficiently large n

$$\mathbf{P}\left[X(n) > cf(n)\right] \leq \frac{1}{n}$$

or

$$\mathbf{P}\left[X(n) \leq cf(n)\right] \geq 1 - \frac{1}{n} \geq \mathbf{P}\left[X(n) < \tilde{x}(n)_{1-\frac{1}{n}}\right]$$

by definition of a quantile. This implies

$$\tilde{x}(n)_{1-\frac{1}{n}} \leq cf(n) = O(f(n))$$

Now Lemma 1 can be applied:

$$\mathbf{E}X(n) = \frac{O(f(n))}{1-\frac{1}{n}} = O(f(n))$$

□

5 Analysis

Algorithm analysis is often facing a dilemma between undue simplification of a problem and an informal treatment lacking mathematical rigor. This is particularly true for parallel algorithms where there is not even an agreed upon model of computation. This paper uses a rather informal style in order to make it possible to discuss complications like routing strategies or bus contention without introducing too much notational overhead. However, the probabilistic part is somewhat more rigorous in order to avoid the many pitfalls of too vague argumentation in the context of probability theory.

Figure 1 shows pseudocode for the algorithm underlying the analysis. All PEs execute the same program with the exception that PE 0 initially gets all the work. Idle PEs poll randomly selected PEs for work and reject requests they receive. In practice, an idle PE will not send a request to itself but for the purpose of the analysis this case is not excluded. Busy PEs cycle between doing work and servicing at most one request. Note that a busy PE will not block if no requests are imminent nor can it be swamped by requests without being able to do "useful" work. In addition, some protocol for termination detection is necessary which is not considered here since it is not a bottleneck if implemented properly.

5.1 Framework of the Analysis

The starting point is the definition of efficiency:

$$E = \frac{w}{nT_{\text{par}}}$$

```
initialize PE 0 with the entire work
WHILE NOT finished DO for all PEs in parallel (asynchronously)
    IF task is empty THEN
        REPEAT
            send a request R to a randomly determined PE
            wait for a reply and reject any incoming requests
        UNTIL R is not rejected
        reinitialize task from incoming message
    WHILE task is not empty DO
        IF there is an incoming request THEN
            split task
            asynchronously send one part to the initiator of the request
        do some work on task
```

Figure 1: Pseudocode for RP

For any $\gamma \in [0, 1/2)$ we can set

$$T_{\text{par}} = T_{<\gamma} + T_{\geq\gamma}$$

where $T_{<\gamma}$ is the length of all time intervals during which less than γn PEs are idle. If we neglect the time to test for a request[2] and assume that γn active PEs are busy servicing requests of the idle PEs[3] we get

$$T_{<\gamma} < \frac{w}{n(1-2\gamma)}$$

since in this time the active PEs can process the entire task.

During $T_{\geq\gamma}$ there will be at least $\frac{\gamma n}{T_{\text{req}}}$ work requests per time unit if T_{req} is the time needed for a work request. Let the random variable $K(n, h(w))$ denote the number of work requests necessary such that every task has been split at least $h(w)$ times. Then,

$$T_{\geq\gamma} \leq \frac{K(n, h(w))T_{\text{req}}}{\gamma n} + g(w)$$

Since after time $\frac{K(n,h(w))T_{\text{req}}}{\gamma n}$ every task is reduced to an atomic size. Now the

[2]If incorporated into the analysis, it would turn out that the test implies an upper bound on the efficiency. However, the test can often be implemented very efficiently and intervals between tests can be made arbitrarily large without affecting the asymptotic scalability. The maximal efficiency can therefore be made as close to 1 as desired.

[3]If routing is entirely done by software, this figure becomes $2\gamma n$ since for each message, up to two PEs at a time can be delayed having to transfer the message (assuming a store-and-forward routing policy).

efficiency can be estimated.

$$E \geq \frac{w}{n\left(\frac{w}{n(1-2\gamma)} + \frac{K(n,h(w))T_{\text{req}}}{\gamma n} + g(w)\right)} = \frac{w}{\frac{w}{(1-2\gamma)} + \frac{K(n,h(w))T_{\text{req}}}{\gamma} + ng(w)} \qquad (3)$$

5.2 The Order of $K(n, h(w))$

Under the reasonable assumption that there is at least a constant number of atomic work units for each PE (i. e. $w = g(w)\Omega(n)$) the asymptotic behavior of $K(n, h(w))$ can be derived.

Lemma 2 *Let the random variable $K_t(n, h(w))$ denote the number of requests necessary to hit a particular task t $h(w)$ times; Then $K_t(n, h(w)) = \tilde{O}(nh(w))$ if $w = g(w)\Omega(n)$.*

Proof: We need to find a c such that for all $\beta \geq 1$ and sufficiently large n

$$P := \mathbf{P}\left[K_t(n, h(w)) > c\beta nh(w)\right] \leq n^{-\beta}$$

or

$$\mathbf{P}\left[\text{after } c\beta nh(w) \text{ requests: } (\#\text{ of requests for } t) < h(w)\right] \leq n^{-\beta}$$

Since the requests are independent and task t is hit with the uniform probability $\frac{1}{n}$, Theorem 2 is applicable. By writing $h(w)$ as $\left(1 - \left(1 - \frac{1}{c\beta}\right)\right)(c\beta nh(w))\frac{1}{n}$ we get

$$P \leq \exp - \left[\left(1 - \frac{1}{c\beta}\right)^2 \frac{c\beta h(w)}{3}\right]$$

Since $w = g(w)\Omega(n)$, $h(w) = \Omega(\log n)$ because even a perfect splitting function would always leave a piece of work not in $O(g(w))$ after less than logarithmically many splits. So, there is a constant $d > 0$ such that $h(w) \geq d \ln n$ for sufficiently large n. Using $\beta \geq 1$ we can further estimate:

$$\begin{aligned} P &\leq \exp - \left[\left(1 - \frac{1}{c}\right)^2 \frac{c\beta d \ln n}{3}\right] \\ &= n^{-\beta\left(1-\frac{1}{c}\right)^2 \frac{cd}{3}} \\ &\leq n^{-\beta} \text{ if } c \geq 1 + \frac{3 + \sqrt{12d+9}}{2d} \end{aligned}$$

□

There are only $O(n)$ tasks, and therefore Theorem 1 allows us to conclude that the asymptotic behavior of $K(n, h(w))$ with high probability is the same as the behavior of $K_t(n, h(w))$:

Corollary 1 *$K(n, h(w)) = O(nh(w))$ with high probability if $w = g(w)\Omega(n)$*

Issuing requests can only decrease the expected number of additional requests necessary to hit all tasks at least $h(w)$ times, i. e.
$\mathbf{E}[K(n,h(w))|K(n,h(w)) > \tilde{x}_\alpha] \leq \tilde{x}_\alpha + \mathbf{E}K(n,h(w))$ for all α. Theorem 3 can be used to get:

Corollary 2 *$K(n,h(w)) = O(nh(w))$ on the average if $w = g(w)\Omega(n)$.*

5.3 Estimating $h(w)$

Splitting a task of size v produces two tasks with sizes αv and $(1-\alpha)v$ where $0 < \alpha \leq 1/2$. If it is guaranteed that α is bounded from below by a positive constant then it is fairly straightforward to show that $h(w) = O(\log w)$ [KA91], making the analysis of RP less involved.

However, this assumption is not always warranted. In depth first tree search for example, a very popular splitting function splits the search tree by distributing the successors of the root-node between the two subtasks. If the degree of tree nodes is bounded by a constant, $h(w)$ is proportional to the height of the tree. If the tree has a sufficiently uniform shape, the height is indeed logarithmic in w. But, there are search algorithms where both the height of the tree and w are polynomial in some input measure [Pea84] and therefore figures like $h(w) \sim \sqrt{w}$ are quite conceivable.

Although there are more sophisticated splitting functions for search trees [KR87], it is an open question in which cases these functions can guarantee that $h(w) = O(\log w)$ (perhaps in some probabilistic sense). But even if this works, the analysis for general $h(w)$ may help to decide whether the additional expense for a more sophisticated splitting function is worth the effort.

Another example for trees of very irregular shape are computation trees induced by functional programs. According to [ABF93], it is quite difficult to come up with a useful splitting function for those trees. The same problem is to be expected for logical programming languages.

5.4 The Request Delay T_{req}

Since there cannot be more messages than idle PEs, there are at most γn messages at a time. The messages have independent randomly determined destinations and have size $O(s(w))$. Analyzing this routing problem is a nontrivial problem by itself. In analogy to [KA91] we will therefore assume that at least packets of constant size can be routed in $O(d(n))$ time for meshes, hypercubes and various multi-stage networks. Then

$$T_{\text{req}} = O(d(n)s(w)) \tag{4}$$

Other networks are limited by their bisection bandwidth. For example, on busses or trees it can only be said that $T_{\text{req}} = O(ns(w))$. The subsequent analysis

works with $T_{\text{req}} = O(d(n)s(w))$. The result for other cases is easy to obtain by substituting the appropriate values.

Note that it may be an overestimation to assume that some fraction of the messages has size $s(w)$. As few as $\Omega(n)$ work transfers may suffice to balance the load. On the other hand, a total of $O(nh(w))$ requests is to be expected. Also, for some settings, the actual length of a message decreases with the actual size of a task whereas we always count the upper bound for a given *initial* problem size. In addition, the delivery of long messages can be accelerated by pipelining (i. e. chopping the message into pieces of constant size). In this case, delivery is possible in time $O(d(n) + s(w))$ when network traffic is low. For Hypercubes, there are randomized algorithms for a similar routing problem which work in time $O(\max\{s(w), \log n\})$ even if all messages have full size [ALMN91].

5.5 The Isoefficiency of RP

In the preceding sections all factors influencing the performance of RP have been estimated. Now it is possible to put the individual pieces together. Using Relation 3, Corollary 1 and 2 and Equation 4 we can conclude that there is a constant c such that for sufficiently large n and w:

$$E \geq \frac{w}{\frac{w}{(1-2\gamma)} + \frac{cnh(w)d(n)s(w)}{\gamma} + ng(w)} \tag{5}$$

both with high probability and on the average if $w = g(w)\Omega(n)$.

An immediate observation is that for $g(w) > ch(w)s(w)d(n)$ the scalability is dominated by the atomic grainsize. In this case it may not be necessary to bother about routing delays, quality of splitting function or message sizes.

If $h(w)s(w) = \Omega(w)$ or $g(w) = \Omega(w)$ we have $\lim_{n\to\infty} E = 0$ according to our estimate. So, for large $h(w)s(w)$ or $g(w)$ RP may not be scalable at all. Indeed, for $h(w) = \Omega(w)$ or $g(w) = \Omega(w)$ the problem contains a sequential component of size $\Omega(w)$ and no scalable parallel algorithm is possible.

For $\max\{h(w)s(w), g(w)\}$ not in $\Omega(w)$ we can choose $\gamma < \frac{1-E}{2}$ and get $\lim_{w\to\infty} E = 1$ i. e. for sufficiently large w any desired efficiency can be achieved. The degree of scalability for RP can be assessed by fixing E and solving for the isoefficiency function $w(n)$. This is now done for two characteristic cases:

Isoefficiency for $h(w)s(w) = O(\log^a w)$, $a \geq 1$, $g(w) = O(\log^b w)$, $b \geq 0$

We are only interested in polynomial $w(n)$ and therefore $h(w)s(w) = O(\log^a n)$ and $g(w) = O(\log^b n)$, i. e. there is a constant c' such that for sufficiently large n

$$E \geq \frac{w}{\frac{w}{(1-2\gamma)} + \frac{cc'nd(n)\log^a n}{\gamma} + nc'log^b n}$$

Solving for w gives:

$$w \leq \frac{cc'n(d(n)\log^a n + \log^b n))}{\frac{1}{E} - \frac{1}{1-2\gamma}}$$

i. e.

$$w(n) = O(n \max\{d(n)\log^a n, \log^b n\}) \tag{6}$$

with high probability and on the average. The case $h(w) = O(\log w)$ and $s(w), g(w) = O(1)$ is proven in [KA91] to be in $O(nd(n)\log^2 n)$ on the average. So the new result is tighter by a factor of $\log n$.

Isoefficiency for $h(w)s(w) = O(w^\alpha)$, $0 < \alpha < 1$, $g(w) = O(w^\beta)$, $0 \leq \beta < 1$

Using a similar discussion as for Equation 6 we get:

$$w(n) = O(\max\{(nd(n))^{\frac{1}{1-\alpha}}, n^{\frac{1}{1-\beta}}\}) \tag{7}$$

with high probability and on the average.

And for the mixed case $h(w) = O(\log w)$, $s(w) = O(w^\alpha)$ with $\alpha \geq 0$:

$$w(n) = O(\max\{(nd(n)\log n)^{\frac{1}{1-\alpha}}, n^{\frac{1}{1-\beta}}\}) \tag{8}$$

with high probability and on the average.

5.6 Lower Bounds

There are four things which have to be done by any scalable parallel program which uses the problem model from Section 2:

1. Some processor has to process a task of size $\Omega(w/n)$.
2. Some processor has to process a task of size $\Omega(g(w))$.
3. If a significant share of the PEs shall be utilized, on any reasonable network[4], some task has to travel a distance in $\Omega(d(n))$ incurring a transmission time of $\Omega(d(n) + s(w))$.
4. Under the assumption that the incore representation of a task is not asymptotically shorter than its representation as a message, a time in $\Omega(s(w)\log n)$ has to be invested to split some task $\Omega(\log n)$ times. Else, a task would exist which is too large to be processed in time $O(w/n)$.

[4] As an example of an "unreasonable" network consider a network of $n/2$ PEs which are fully connected plus a "tail" of another $n/2$ PEs arranged as a linear array.

As a consequence, the parallel execution time T_{par} is in $\Omega(\max\{w/n, d(n)+s(w), s(w)\log n, g(w)\}) = \Omega(\max\{w/n, d(n), s(w)\log n, g(w)\})$ and therefore there is a constant c such that for sufficiently large n and w:

$$E \leq \frac{w}{nc\max\{w/n, d(n), s(w)\log n, g(w)\}}$$

In analogy to the discussion in Section 5.5 this relation can be used to derive lower bounds for the isoefficiency. For $s(w) = \Omega(\log^a w)$ and $g(w) = \Omega(\log^b w)$ with $a, b \geq 0$

$$w(n) = \Omega(n\max\{d(n), \log^a n, \log^b n\}) \tag{9}$$

and for $s(w) = \Omega(w^\alpha)$, $g(w) = \Omega(w^\beta)$ with $0 \leq \alpha < 1$, $0 \leq \beta < 1$

$$w(n) = \Omega(\max\{nd(n), (n\log n)^{\frac{1}{1-\alpha}}, n^{\frac{1}{1-\beta}}\}) \tag{10}$$

By comparing equation 9 and 10 with equations 6 and 8 respectively we see that for $g(w) = \Omega(h(w)s(w)d(n))$ or $d(n) = O(1)$ and $h(w) = O(\log w)$ the scalability of RP is asymptotically optimal. The assumption $d(n) = O(1)$ is certainly unrealistic for large n. However, on many contemporary machines, message startup times dominate the time for delivery for any practical value of n and a constant diameter may be a good approximation.

How Tight is the Analysis?

A discussion similar to the derivation of a general lower bound can be used to derive a lower bound for the efficiency of RP itself. In order to make all tasks sufficiently small, there has to be some chain of splits of length $\Omega(\log n)$. The crucial difference is that in RP for each split there has to be a corresponding request. Assuming that some fraction[5] of the corresponding requests has to traverse a distance of $\Omega(d(n))$, $\Omega(d(n)\log n)$ is a lower bound for the parallel execution time. The corresponding bound for the isoefficiency is:

$$w(n) = \Omega(nd(n)\log n) \tag{11}$$

So, for the case of a good splitting function ($h(w) = O(\log w)$) and bounded message lengths ($s(w) = O(1)$) the analysis from Section 5.5 (Equation 6) turns out to be tight.

6 Initialization Methods

When the basic RP algorithm is started, only one PE is active and it takes some time until PE utilization is satisfactory. During this startup phase, many

[5]On most network types this will be true with high probability.

work requests fail. This problem can be avoided by using a more sophisticated initialization method, first described in [EDH80], and called *selective initialization* in [Hen93]: First, the entire task is broadcast to all PEs. Then, the task is split repeatedly and the bits of the PE index are used to decide which subtask is retained. After $\log n$ splits each PE is guaranteed to have a different piece of work. The scheme can be generalized for cases where n is not a power of 2.

Figure 2 shows the effect of selective initialization on a simulated run of RP for a relatively small problem on a machine with 16384-PEs using a splitting function that splits tasks in the ratio $1:2$.

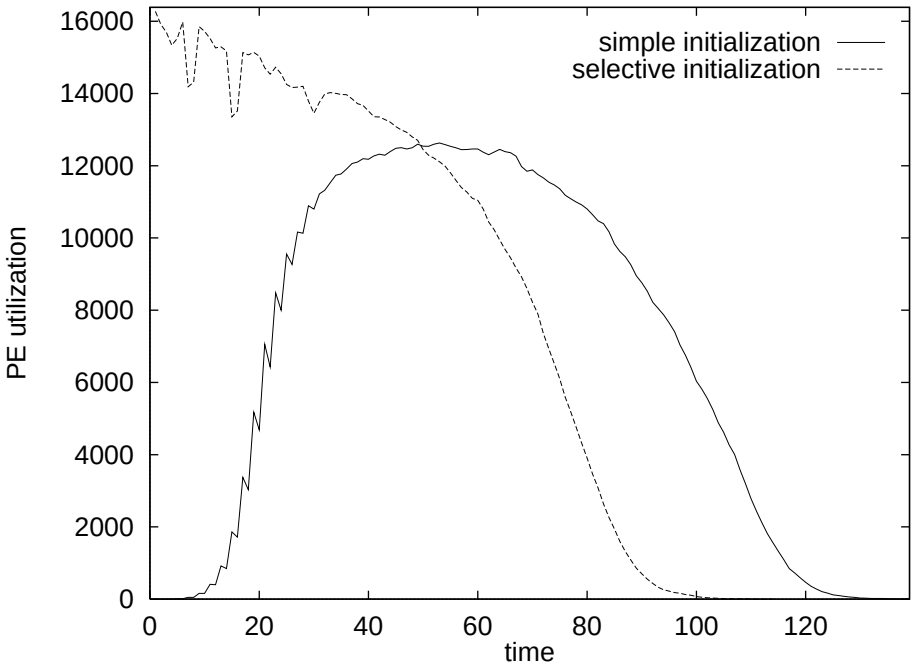

Figure 2: PE utilisation over time with and without selective initialization.

It can be observed that selective initialization effectively eliminates the start-up phase of low PE-utilization observed for basic RP. However, the tail of low PE-utilization towards the end remains unaffected.

The impact made by initialization depends on the quality of the splitting function. If it is perfect, i. e. it always splits a task in equal halfs, no dynamic load balancing is needed at all. But in general, the $\log n$ splits performed by initialization go only part of the way. It is to be expected that the subsequent RP phase takes asymptotically as long as basic RP.

A point in favor of initialization schemes is that for some applications the work-load does not slowly "dry out" as in our model, but eventually one PE finds a problem solution and immediately broadcasts a termination message. In this

case, the startup phase may be the main source of overhead.

7 Conclusions and Future Work

The simple randomized dynamic load balancing algorithm Random Polling has often proved useful in practice. This paper helps to explain its performance using quite general assumptions about the application and the underlying parallel machine. For bounded message sizes and good splitting functions it derives new tight scalability bounds. If the communication overhead is dominated by the atomic grainsize of the problem or if the time needed for the delivery of a message depends only on the problem size, then RP is asymptotically optimal. Figure 3 summarizes the results on the isoefficiency function of RP for some characteristic cases. A simple initialization scheme which involves only a single broadcast can be used to eliminate a period of low PE utilization during startup.

network type $\downarrow$	$d(n)$ $\downarrow$	$\log w$ 1	$\log w$ $\log^b w$	$\log w$ w^b	$w^a \leftarrow h(w)$ $w^b \leftarrow s(w)$
crossbar	1	$n \log n$	$n \log^{1+b} n$	$(n \log n)^{\frac{1}{1-b}}$	$n^{\frac{1}{1-a-b}}$
fat tree, ...	$\log n$	$n \log^2 n$	$n \log^{2+b} n$	$(n \log^2 n)^{\frac{1}{1-b}}$	$(n \log n)^{\frac{1}{1-a-b}}$
3D mesh/torus	$\sqrt[3]{n}$	$n^{\frac{4}{3}} \log n$	$n^{\frac{4}{3}} \log^{1+b} n$	$(n^{\frac{4}{3}} \log n)^{\frac{1}{1-b}}$	$n^{\frac{4}{3(1-a-b)}}$
2D mesh/torus	$\sqrt{n}$	$n^{\frac{3}{2}} \log n$	$n^{\frac{3}{2}} \log^{1+b} n$	$(n^{\frac{3}{2}} \log n)^{\frac{1}{1-b}}$	$n^{\frac{3}{2(1-a-b)}}$
ring, bus, tree	n	$n^2 \log n$	$n^2 \log^{1+b} n$	$(n^2 \log n)^{\frac{1}{1-b}}$	$n^{\frac{2}{1-a-b}}$

Figure 3: Isoefficiency function of Random Polling if $g(w)$ is dominated by $h(w)s(w)d(n)$; $O(\cdot)$ always implicit; $\dddot{=}$: optimal; $\|\cdot\|$: tight bound.

An interesting methodological experience is that it may be useful to derive the average case behavior of an algorithm indirectly by first investigating the asymptotic behavior with high probability. One reason for this may be the lack of an average case equivalent to Theorem 1 which makes it possible to reduce the behavior of many parallel processes to the sequential case. By accepting some complications for the derivation it is possible to incorporate many details of the application and the parallel machine into the analysis.

An open question in the analysis of RP is the influence of message sizes on performance: How many tasks are actually transferred? Would a routing strategy involving pipelining be able to transmit long messages faster or would it be hobbled by network contention? What, if the length of a task representation is strongly dependent on its actual size? The dominating open question is, how a practicable load balancing scheme might look like that is asymptotically more

efficient than Random Polling for the type of tree structured computations considered here.

References

[ABF93] G. Aharoni, A. Barak, and Y. Farber. An adaptive granularity control algorithm for the parallel execution of functional programs. *New Generation Computing Systems*, 9:163–174, 1993.

[ALMN91] Aiello, Leighton, Maggs, and Newman. Fast algorithms for bit-serial routing on a hypercube. *Mathematical Systems Theory*, 24:253–271, 1991. Appeared also in 2nd Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA 90).

[Che52] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Mathematical Statistics*, 23:493–507, 1952.

[EDH80] O. I. El-Dessouki and W. H. Huen. Distributed enumeration on between computers. *IEEE Transactions on Computers*, C-29(9):818–825, September 1980.

[FM87] R. Finkel and U. Manber. DIB— A distributed implementation of backtracking. *ACM Trans. Prog. Lang. and Syst.*, 9(2):235–256, April 1987.

[FMM91] Rainer Feldmann, Peter Mysliwietz, and Burkhard Monien. Distributed game tree search on a massively parallel system. In Th. Ottmann B. Monien, editor, *Data structures and efficient algorithms: Final report on the DFG special joint initiative*, volume LNCS 594, pages 270–288. Springer-Verlag, September 1991.

[Hen93] D. Henrich. Initialization of parallel branch-and-bound algorithms. In *Proceedings of PPAI-93*, 1993.

[KA91] V. Kumar and G. Y. Ananth. Scalable load balancing techniques for parallel computers. Technical Report TR 91-55, University of Minnesota, 1991.

[KR87] V. Kumar and V. N. Rao. Parallel depth first search. Part I. *International Journal of Parallel Programming*, 16(6):470–499, 1987.

[Kum90] V. Kumar. Scalable parallel formulations of depth-first search. In W. Kumar, editor, *Parallel Algorithms for Machine Intelligence and Vision*. Springer, 1990.

[KZ93] R. M. Karp and Y. Zhang. Parallel algorithms for backtrack search and branch-and-bound. *Journal of the ACM*, 40(3):765–789, 1993.

[L+89] T. Leighton et al. Dynamic tree embeddings in butterflies and hypercubes. In *ACM Parallel Processing Symposium*, pages 224–234, 1989.

[Lei92] T. Leighton. *Introduction to Parallel Algorithms and Architectures.* Morgan Kaufmann, 1992.

[Pea84] Judea Pearl. *Heuristics.* Addison Wesley, 1984.

[PFK93] C. Powley, C. Ferguson, and R. E. Korf. Depth-fist heuristic search on a SIMD machine. *Artificial Intelligence*, 60:199–242, 1993.

[Raj92] S. Rajasekaran. Randomized algorithms for packet routing on the mesh. In L. Kronsjö and D. Shumsheruddin, editors, *Advances in Parallel Algorithms*, pages 277–301. Blackwell, 1992.

[Ran94] A. Ranade. Optimal speedup for backtrack search on a butterfly network. *Mathematical Systems Theory*, pages 85–101, 1994.

[San94a] P. Sanders. Massively parallel search for transition-tables of polyautomata. In *Parcella 94, VI. International Workshop on Parallel Proccessing by Cellular Automata and Arrays (submitted)*, Potsdam, 1994.

[San94b] P. Sanders. Portable parallele Baumsuchverfahren: Entwurf einer effizienten Bibliothek. In *Transputer Anwender Treffen (submitted)*, Aachen, 1994.